

PATENT APPLICATION TRANSMITTAL LETTER

To Assistant Commissioner of Patents
BOX PATENT APPLICATION
Washington, DC 20231



Transmitted herewith for filing of the patent application of:

Inventor(s): Anthony P. GALLUSCIO, Indialantic, FL
William L. HOLT, Coco Beach, FL
Douglas M. DYER, Indialantic, FL
Albert T. MONTROY, Melbourne, FL

for HIGH SPEED INTERPROCESS COMMUNICATION.

are the following:

- ☒ Specification, including Abstract
☐ 5 Sheets of drawing (3 sets)
☒ Declaration and Power of Attorney (Unexecuted)
☒ Other: 2 postcards

CLAIMS AS FILED

FOR	NO. FILED	NO. EXTRA
Basic Fee		
Total Claims	-20- 18	0
Indep Claims	-3- 3	0
Multiple dependent claim present No		

Small Entity

RATE	FEE
	\$ 380
x \$ 9 =	\$
x \$ 39 =	\$ 0
x \$130 =	\$ 0
TOTAL	\$380

Other than a Small Entity

RATE	FEE
	\$
x \$ 18 =	\$
x \$ 78 =	\$
x \$260 =	\$
TOTAL	\$

If the difference in Col. 1 is less than zero, enter "0" in Col. 2

assignment recordal fee

\$

___ Please charge my Deposit Account No. 17-0055 in the amount of \$ _____.

___ A check in the amount of _____ is enclosed.

___ The Commissioner is hereby authorized to charge payment of the following fees associated with this communication or credit any overpayment to Deposit Account No. 17-0055. A duplicate of this sheet is enclosed.

___ Any additional filing fees required under 37 C.F.R. 1.16.

___ Any patent application processing fees under 37 C.F.R. 1.17.

☒ No fee enclosed. No fee is authorized.

11/1/99
Date

Robert J. Sacco
Registration No. 35,667

"EXPRESS MAIL" MAILING LABEL NO.: EE444322314US

HIGH SPEED INTERPROCESS COMMUNICATION

Inventor(s): Anthony P. Galluscio
William L. Holt
Douglas M. Dyer
Albert T. Montroy

Exigent International, Inc.

Exigent Docket No.: P-044

CROSS REFERENCE TO RELATED APPLICATIONS

(Not Applicable)

**STATEMENT REGARDING FEDERALLY SPONSORED RESEARCH OR
DEVELOPMENT**

(Not Applicable)

BACKGROUND OF THE INVENTION**Technical Field**

This invention relates to the field of network interprocess communications and more particularly to a system and method for high speed interprocess communications.

Description of the Related Art

Interprocess communications (IPC) includes both process synchronization and messaging. Messaging, or message passing, can be accomplished using pipes, sockets, or message queues. Pipes provide a byte stream mechanism for transferring data between processes. System calls, for example read and write calls, provide the underlying mechanisms for reading and writing data to a pipe. As the writing process writes to a pipe, bytes are copied from the sending process into a shared data page. Subsequently, when the reading process reads, the bytes are copied out of the shared data page to the destination process. Sending a message using a pipe consumes a minimum of two memory copies moving a total of $2n$ bytes, where n is the number of bytes in the message. Sockets provide an abstraction mechanism for application programs which can simplify access to communications protocols. Although network communications provide the primary impetus for sockets, sockets may be used for IPC, as well. Still, the transmission of data between processes using a socket can consume time necessary to perform

the abstraction overhead, two system calls, for example `readv()` and `writew()`, and a minimum of two memory copies moving, in total, $2n$ bytes. A message queue is an IPC mechanism that is typically managed by an operating system kernel. Typically, the implementation of a message queue is hidden.

5 Traditional IPC mechanisms can provide IPC between two processes using virtual memory in a shared memory space contained within an operating system kernel. The use of virtual memory implies that, although all processes share the same physical memory space, each process can map a region of the shared memory space differently from other processes. Thus, data residing at one address
10 in the shared memory space can differ physically from the data residing at the same address into the same shared memory space as interpreted by the memory map of a different process.

In traditional IPC, a first process can copy n bytes of data from user memory space into a shared memory space in the operating system kernel. Subsequently,
15 using a system call to the operating system kernel, a second process can copy the same n bytes of data from the shared memory space into the user memory space. Therefore, traditional IPC mechanisms require a minimum overhead of $2n$ byte copies to communicate n bytes of data between the two processes.

20 In addition, all methods of traditional IPC require some type of interaction with the operating system kernel. In particular, traditional IPC mechanisms require a minimum of two system calls to the operating system kernel. Moving data in and out of an operating system kernel can include some risk. Specifically, not only must each process move n bytes of data, but each process risks losing CPU control upon invoking the system call required to read or write the data, respectively.

25 Analogously, computer scientists have recognized the unnecessary expense of passing a message from one process to another. In fact, legacy third-generation programming languages which provide dynamic memory allocation, for example Fortran, inefficiently pass data between processes by copying the data stored in

one region of memory, and storing the data in a different region of memory.

Subsequently, the recipient function can process the data before returning a copy of the same using the same mechanism. Recognizing the inefficiencies of this type of message passing, computer scientists have adopted pointer passing as an

alternative to data passing when messaging a process. In pointer passing, a recipient process receives only an address of a location in memory of the message data. Subsequently, the recipient can manipulate the data at the passed address, in place, without the need for excessive data copies.

Still, in third-generation languages which have adopted pointer passing, for

example C or C++, the communicating processes ultimately share one memory mapping of a shared memory space for passing data. In fact, in the absence of a single memory map of shared memory space, present methods of pointer passing become unworkable because data residing at an address in one memory space is not equivalent to the data residing at the same address in another memory space.

Therefore, traditional pointer passing cannot be used to resolve the inefficiencies of traditional IPC in which different processes have different memory maps of a shared region of user memory by virtue of the virtual memory scheme associated with network IPC.

In view of the inefficiencies of traditional IPC, traditional mechanisms for IPC are not suitable for real time command and control systems which can require fail-safe and extremely fast conveyancing of information between processes. For example, copying data can be expensive in terms of processor overhead and time delay. In addition, moving data in and out of an operating system kernel can include some risk. Thus, present IPC mechanisms do not provide the level of service required for real-time applications.

SUMMARY OF THE INVENTION

In a preferred embodiment, a method for high speed interprocess communications can comprise four steps. Initially, first and second processes can be attached to a message buffer in a shared region of user memory (RAM).

Moreover, message lists corresponding to each of the processes can be established in the shared region. In particular, the attaching step can comprise the steps of: detecting a previously created shared region of user RAM; if a shared region of RAM is not detected, creating and configuring a shared region of user memory for storing accumulated data; and, attaching to the created and configured shared region of RAM. In a preferred embodiment, the attaching step comprises the step of attaching the first and second processes to a message buffer in a shared region of RAM exclusive of operating system kernel space. Message data from the first process can be accumulated in a location in the message buffer.

Advantageously, the message list can be implemented as a message queue using the common data structure, "queue". As a result, subsequent to the accumulating step, a memory offset corresponding to the location in the message buffer can be added to the message queue of the second process. The adding step can comprise the steps of: retrieving a memory offset in the message buffer corresponding to the location of message data accumulated by the first process; and, inserting the memory offset in the message queue corresponding to the second process. Moreover, the inserting step can comprise the step of atomically assigning the memory offset to an integer location in the message queue corresponding to the second process.

Finally, the accumulated message data at the location corresponding to the memory offset can be processed in the second process. The processing step can comprise the steps of: identifying a memory offset in the message list corresponding to the second process; processing in the second process message data stored at a location in the message buffer corresponding to the memory offset;

and, releasing the message buffer. Consequently, the accumulated message data is transferred from the first process to the second process with minimal data transfer overhead.

Viewed from a system architecture standpoint, a method for configuring high speed interprocess communications between first and second processes can include several steps. Initially, the method can include disposing a message buffer in a shared region of RAM shared between first and second processes. In particular, the disposing step can comprise the steps of: creating and configuring a message buffer in a shared region of RAM exclusive of operating system kernel space; and, creating a message list in the shared region for each process, whereby the message list can store memory offsets of message data stored in the message buffer.

The inventive method can include the step of accumulating message data from the first process in a location in the message buffer and adding the memory offset to a message list corresponding to the second process. Advantageously, the message list can be implemented as a message queue. In consequence, the adding step can comprise the steps of: retrieving a memory offset in the message buffer, the memory offset corresponding to the location of message data accumulated by the first process; and, inserting the memory offset in the message queue corresponding to the second process. Moreover, the inserting step can comprise the step of atomically assigning the memory offset to an integer location in the message queue corresponding to the second process.

Finally, the method can include processing in the second process the accumulated message data stored in the message buffer at a location corresponding to the memory offset. In particular, the processing step comprises the steps of: identifying a memory offset in the message list corresponding to the second process; using in the second process accumulated message data at a location in the message buffer corresponding to the memory offset; and, releasing the message

buffer. Thus, as a result of the inventive method, the accumulated message data can be transferred from the first process to the second process with minimal data transfer overhead.

6572-14

BRIEF DESCRIPTION OF THE DRAWINGS

There are presently shown in the drawings embodiments which are presently preferred, it being understood, however, that the invention is not limited to the precise arrangements and instrumentalities shown.

5 Fig. 1 is a schematic illustration of a traditional IPC architecture.

Fig. 2 is a schematic illustration of a high-speed IPC architecture in accordance with the inventive arrangements.

Fig. 3 is a process diagram showing the passing of data using a high-speed IPC architecture in accordance with the inventive arrangements.

10 Fig. 4 is a schematic representation of a memory offset.

Fig. 5 is a flow chart illustrating an algorithm for high-speed IPC.

DETAILED DESCRIPTION OF THE INVENTION

The traditional interprocess communication (IPC) architecture includes drawbacks which preclude the use of traditional IPC mechanisms in real time command and control systems which can require fail-safe and extremely fast conveyancing of information between processes. Figure 1 illustrates the commonality between the three traditional mechanisms for IPC. As shown in the figure, two processes 1, 2 communicate using a shared memory space 5 contained within an operating system kernel 4. In particular, process 1 can copy n bytes of data 6 from a user memory space 3 into a shared memory space 5 in the operating system kernel 4. Subsequently, using a system call, process 2 can copy the same n bytes of data 6 from the shared memory space 5 in the operating system kernel 4 into user memory space 3. Therefore, Figure 1 shows a minimum overhead of two system calls and $2n$ byte copies to communicate n bytes of data between two processes.

In contrast, a method for high speed IPC can provide extremely fast IPC both by communicating message data in a shared region of random access memory (RAM) external to the operating system kernel and by limiting the movement of the data. Processes are notified of the location of the message data rather than actually receiving a copy of the message data. The recipient process subsequently can read or manipulate (process) the message data in place. As a result, the number of data copies necessary for high speed IPC is minimized. Notably, a method for high speed IPC utilizes a message list in a message buffer for storing a memory offsets set by atomic assignment. Each memory offset can denote a location in the shared region of RAM where a process attached to the shared region can manipulate data stored therein. Specifically, the message list can be implemented using the common data structure, "queue". When one process messages another, the process need only insert a memory offset to the message data in the recipient's message queue.

Figure 2 provides a high-level perspective of the relationships between the message data 10, the operating system kernel 14, and two processes 11, 12 using high speed IPC. From Figure 2, it will be apparent to one skilled in the art that the message data 10 resides in a shared region of RAM 15 common to both processes 11, 12. Still, one skilled in the art will further recognize that although the shared region of RAM is common to both processes 11, 12, each process 11, 12 can maintain a virtual memory page therein. That is, each process 11, 12 can maintain a different and distinct memory map of the shared region of user RAM 15. However, unlike prior art network IPC where processes do not reconcile differing memory maps of shared RAM, in the preferred embodiment, both processes 11, 12 can reconcile each other's memory mapping into the shared region of RAM 15 by communicating to one another the location of data in the shared RAM relative to a commonly known address.

Capitalizing on this reconciliation, high speed IPC permits the use of the shared region of RAM 15 despite differing memory maps among the processes 11, 12. As a result, in the preferred embodiment, message passing does not require storing message data 10 in operating system kernel space 14. Therefore, in the preferred embodiment, system calls are not required to write and read the data 10. Thus, the elevated risk associated with utilizing operating system kernel space 14 is eliminated. Specifically, the inventive method avoids the risk of a process losing CPU control upon invoking the system call required to read or write the data 10. Hence, the inventive IPC mechanism can provide the level of service required for real-time applications.

Figure 3 illustrates an exemplary conveyance of data 30 between two processes 21, 22 using the inventive method for high speed IPC. Notwithstanding, the invention is not limited in this regard. Rather, the invention can include more than two processes communicating through a shared message store. The conveyance consists of four essential steps. Initially, a first process 22 and a

second process 23 can attach to a small message buffer 25 from a configured pool of message buffers 24 in a shared region of RAM. In the preferred embodiment, the pool of messages buffers 24 can include small 25, medium 26 and large buffers 27. However, the invention is not limited in this regard. Rather, any number or
5 type of message buffers will suffice for operation of the present invention. For instance, a "first-fit" allocation strategy, a "best-fit" strategy, or an "approximate-fit" strategy can also suffice. Still, the preferred combination of all buffers and the management thereof can optimize memory utilization while reducing the cost of memory management.

10 The first process 22 can accumulate message data 30 in a location in the small message buffer 25. Subsequently, the first process 22 can notify a second process 21 of the location of the data 30 in the small message buffer by adding the location of the data 30 into a message list. Specifically, the first process 22 can insert a memory offset 29 of the message data 30 into a message queue 23
15 associated with the second process 21. As shown in Figure 4, a memory offset B represents the number of bytes C from the beginning A of a buffer D, in which data E can be located. In consequence of using memory offsets, rather than absolute addresses (pointers), two processes can reference a single piece of data in a common region of RAM, despite having different memory maps of the memory
20 region. Hence, although the first process 22 and the second process 21 may have differing memory maps of the small message buffer 25, the memory offset 29 can indicate to each process 21, 22 the number of bytes from a common address of the small message buffer 25 in which the message data 30 can be located.

Message queues 23, 28 preferably are created in the shared region of RAM.
25 Each message queue 23, 28 is a list of messages which can be represented by the common data structure, "queue", which, in the preferred embodiment, can handle integer values in a first-in-first-out (FIFO) order. Each message queue 23, 28, alternatively referred to as an "inbox", can contain an administrative area having

variables for administering the queue of integer offsets. Those variables may include variables for tracking the position of the front and rear elements of the queue and the queue size.

The first process 22 can access the message queue 23 of the second process 21 by addressing the message queue 23 by name. The first process 22 can either have a priori knowledge of the name of the message queue 23, or the first process 22 can rely on a naming service. Specifically, the first process 22 can cross-reference in a naming service the process identification number corresponding to the second process 21 with the location of the message queue 23 of the second process 21. The naming service can be as simple as a file that contains names of message queues mapped to processes. The naming of message queues can depend on the nature of the specific operating system. For instance, in the Windows NT operating system, the operating system names the message queue. In contrast, the Unix operating system uses integer identifiers to identify a message queue.

The memory offset 29 can be logically inserted in the message queue 23 of the second process 21, but advantageously, because the memory offset 29 can be internally represented as an integer, the memory offset 29 can be physically assigned to a data member in a node in the message queue 23 using a simple integer assignment available, for instance, in the C, C++ or Java programming languages. The mechanism for assignment can vary depending on the implementation of the queue data structure. However, as an example, the first process 22 can calculate the address of the first element in the message queue 23, and can make an atomic assignment of the memory offset 29 to that address.

Specifically, in C-syntax, the physical assignment can consist of `"*(inbox_address + front_of_queue) = offset_of_message_data"`. This C-style statement can atomically assign the memory offset 29 of the message data 30 to the address of the first element in message queue 23. The assignment can be

atomic in that a single instruction is required, e.g. "newValue = 5". In the case of an atomic assignment, the entire integer memory offset 29 can be written to the message queue 23 using the single instruction. The atomic assignment can be contrasted with the case of copying a data message using a memory copy, such as "memcpy", which performs an assignment for each byte in the data message.

The front_of_queue variable can be stored at a pre-determined location, as specified by the message queue structure in the beginning of the small message buffer 25. Still, one skilled in the art will recognize that the message queue 23 needn't be stored in the small message buffer 25. Rather, the message queue 23 can be stored in another message buffer, into which access can be provided using any of the traditional IPC techniques. Alternatively and advantageously, access to the message buffer could occur using high speed IPC.

Finally, the second process 21 can identify the memory offset 29 placed in the corresponding message queue 23. Specifically, the second process 21 can poll the message queue 23 waiting for a new memory offset 29 to arrive. Alternatively, the first process 22 can signal the second process 21 that new message data 30 has arrived. Either mechanism can be acceptable depending upon specific application requirements.

Having identified the memory offset 29, the second process 21 can manipulate the accumulated message data 30 in place in the small message buffer 25 corresponding to the memory location denoted by the memory offset 29. The second process 21 can use the accumulated data 30 in accordance with the unique data requirements of the second process 21. When finished, the second process 21 can release the small message buffer 25 using conventional memory management techniques.

Figure 5 is a flow chart describing a method for high speed IPC. The flow chart depicts a single process which can communicate with another process using the inventive method. As shown in the drawings, the method begins in step 100

where a process can attempt to attach to a message buffer in a shared region of RAM, exclusive of the operating system kernel. One skilled in the art will recognize that each attempt to attach to the message buffer will include moderation by a locking mechanism in order to prevent a logic race condition. One such example of a locking mechanism is a mutex which allows an atomic check and set of a variable that protects a shared region. If one process has the mutex, other processes are blocked from accessing the shared region until the mutex is released.

In decision step 102, if the process is the first activated process in the system, in step 104, a message buffer in the shared region of RAM is created and, in step 106, configured. Preferably, the process creating the shared region of RAM obtains a mutex and releases the mutex only when the shared region is created and configured. The release of the mutex acts as notification to other interested processes that the shared region of RAM is ready for use.

Configuring the shared region of RAM can include naming the shared region, initializing the shared region variables in an administrative area, and sizing the buffer pools. Notably, the shared region may be configured using a stored configuration that is merely retrieved by the process and applied to the shared region.

Whether the process creates and configures a new shared region or attaches to a previously created shared region, in step 108, the process can create a message queue in the shared region corresponding to the process. In particular, the message queue can be used to store incoming memory offsets, placed in the message queue by other processes. Having attached to a message buffer and created a message queue, in step 110, the process can perform normal intraprocess operations until a need for IPC arises, either where the process is a recipient or sender of a message, as determined in decision step 112.

If the process is a first process attempting to transmit data to a second process, the first process, in step 122 can obtain a memory offset to free memory

space in the message buffer. One skilled in the art will recognize that obtaining a memory offset to free memory requires the use of a memory management mechanism for allocating buffers in a shared region of user memory. Still, one skilled in the art will further recognize the widespread availability of memory management mechanisms suitable for accomplishing the same. For example, just as the "malloc()" function included as part of the ANSI C standard library can abstract the details of memory management, a buffer pool allocator for high speed IPC can abstract the details of managing memory offsets into the message buffer. The buffer pool allocator for high speed IPC can be implemented using techniques well known in the art and thoroughly documented in Kernighan and Ritchie, "The C Programming Language: 2nd Edition", pp. 185-189, incorporated herein by reference.

Subsequently, in step 124, the first process can accumulate message data for the benefit of the second process with the writing beginning at the location corresponding to the memory offset. When finished accumulating the message data in the message buffer, in steps 126 and 128, the first process can place the memory offset in the message queue corresponding to the second process. Significantly, the placement of the memory offset can be an atomic assignment to an integer location in the shared region of RAM. The act of placing the memory offset in the message queue is tantamount to notifying the second process of an attempt at IPC.

Correspondingly, if the process is a second process receiving a request for IPC from a first process, in step 114, the second process can identify a memory offset in the message queue corresponding to the second process. In step 116, the second process can retrieve the memory offset, and in step 118, the second process can use the memory offset to access the data accumulated by the first process at an appropriate location in the message buffer. Significantly, because the accumulated data is stored in a shared region of user memory, it is not necessary

for the second process to copy the accumulated data to a different memory space. Rather, in step 120, when finished using the data, the second process need only release the buffer using the above-identified buffer pool allocator.

As illustrated in Figure 4, the significant differences between the inventive method and traditional IPC mechanisms include the present method's use of a shared region of RAM to store accumulated data. As a result of the use of the shared region, the inventive method does not require operating system calls to write and read accumulated data. In addition, because the present method uses a shared region of RAM instead of a memory region in the operating system kernel, the reconfiguration of the shared region does not require the rebooting of the operating system. Finally, high speed IPC provides a faster and safer mechanism for IPC in that the overhead associated with IPC is minimized from two system calls and $2n$ bytes of data movement to a minimal n bytes of data movement.

CLAIMS

1. A method for high speed interprocess communications comprising the steps of:

attaching first and second processes to a message buffer in a shared region of random access memory (RAM) exclusive of operating system kernel space, each said process having a message list;

accumulating message data from said first process in a location in said message buffer;

adding to said message list of said second process a memory offset corresponding to said location in said message buffer; and,

processing in said second process said accumulated data at said location corresponding to said offset,

whereby said accumulated message data is transferred from said first process to said second process with minimal data transfer overhead.

2. The method according to claim 1, wherein the attaching step comprises the steps of:

detecting a previously created shared region of RAM;

if a shared region of RAM is not detected, creating and configuring a shared region of RAM for storing accumulated data; and,

attaching said first and second processes to said shared region.

3. The method according to claim 1, wherein said message list is a message queue.

4. The method according to claim 3, wherein the adding step comprises the steps of:

retrieving a memory offset in said message buffer corresponding to said

4 location of data accumulated by said first process; and,
5 inserting said memory offset in said message queue corresponding to said
6 second process.

1 5. The method according to claim 4, wherein the inserting step comprises the
2 step of atomically assigning said memory offset to an integer location in said
3 message queue corresponding to said second process.

1 6. The method according to claim 1, wherein the processing step comprises the
2 steps of:

3 identifying a memory offset in said message list corresponding to said second
4 process;

5 processing in said second process message data stored at a location in said
6 message buffer corresponding to said memory offset; and,
7 releasing said message buffer.

1 7. A method for configuring high speed interprocess communications between
2 first and second processes comprising the steps of:

3 disposing a message buffer in a shared region of random access memory
4 (RAM) shared between said first and second processes;

5 accumulating message data from said first process in a location in said
6 message buffer;

7 adding to a message list corresponding to said second process a memory
8 offset corresponding to said location in said message buffer; and,

9 processing in said second process said accumulated message data stored in
10 said message buffer at a location corresponding to said offset,

11 whereby said accumulated message data is transferred from said first
12 process to said second process with minimal data transfer overhead.

13 8. The method according to claim 7, wherein the disposing step comprises the
14 steps of:

15 creating and configuring a message buffer in a shared region of RAM
16 exclusive of operating system kernel space; and,
17 creating a message list in said shared region for each said process,
18 whereby said message list can store memory offsets of message data stored
19 in said message buffer.

1 9. The method according to claim 7, wherein said message list is a message
2 queue.

10. The method according to claim 9, wherein the adding step comprises the
steps of:

retrieving a memory offset in said message buffer, said memory offset
corresponding to said location of said message data accumulated by said first
process; and,
inserting said memory offset in said message queue corresponding to said
second process.

11. The method according to claim 10, wherein the inserting step comprises the
step of atomically assigning said memory offset to an integer location in said
message queue corresponding to said second process.

12. The method according to claim 7, wherein the processing step comprises the
steps of:

identifying a memory offset in said message list corresponding to said second
process;
processing in said second process said accumulated message data at a

6 location in said message buffer corresponding to said memory offset; and,
7 releasing said message buffer.

1 13. A computer apparatus programmed with a set of instructions stored in a
2 fixed medium for high speed interprocess communications, said programmed
3 computer apparatus comprising:

4 means for attaching first and second processes to a message buffer in a
5 shared region of random access memory (RAM) exclusive of operating system
6 kernel space, each said process having a message list;

7 means for accumulating message data from said first process in a location in
8 said message buffer;

9 means for adding to said message list of said second process a memory
10 offset corresponding to said location in said message buffer; and,

11 means for processing in said second process said accumulated data at said
12 location corresponding to said offset.

13 14. The computer apparatus of claim 13, wherein the attaching means
14 comprises:

15 means for detecting a previously created shared region of RAM; and,

16 means for creating and configuring a shared region in RAM for storing
17 accumulated data if a previously created shared region of RAM is not detected by
18 said detecting means.

1 15. The computer apparatus according to claim 13, wherein said message list is
2 a message queue.

1 16. The computer apparatus according to claim 15, wherein the adding means
2 comprises:

means for retrieving a memory offset in said message buffer corresponding to said location of data accumulated by said first process; and,

means for inserting said memory offset in said message queue corresponding to said second process.

17. The computer apparatus according to claim 16, wherein the inserting means comprises means for atomically assigning said memory offset to an integer location in said message queue corresponding to said second process.

18. The computer apparatus according to claim 13, wherein the processing means comprises:

means for identifying a memory offset in said message list corresponding to said second process;

means for using in said second process message data at a location in said message buffer corresponding to said memory offset; and,

means for releasing said message buffer.

ABSTRACT

A method for high speed interprocess communications comprises four steps. Initially, first and second processes can be attached to a message buffer in a shared region of user memory. In addition, each process can have a corresponding message queue. In a preferred embodiment, the attaching step comprises the step of attaching first and second processes to a message buffer in a shared region of user memory exclusive of operating system kernel space. Second, message data from the first process can be accumulated in a location in the message buffer. Third, a memory offset corresponding to the location in the message buffer can be placed in the message queue of the second process. Finally, the accumulated data at the location corresponding to the offset can be used in the second process. Consequently, the accumulated message data is transferred from the first process to the second process with minimal data transfer overhead.

10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000
1001
1002
1003
1004
1005
1006
1007
1008
1009
1010
1011
1012
1013
1014
1015
1016
1017
1018
1019
1020
1021
1022
1023
1024
1025
1026
1027
1028
1029
1030
1031
1032
1033
1034
1035
1036
1037
1038
1039
1040
1041
1042
1043
1044
1045
1046
1047
1048
1049
1050
1051
1052
1053
1054
1055
1056
1057
1058
1059
1060
1061
1062
1063
1064
1065
1066
1067
1068
1069
1070
1071
1072
1073
1074
1075
1076
1077
1078
1079
1080
1081
1082
1083
1084
1085
1086
1087
1088
1089
1090
1091
1092
1093
1094
1095
1096
1097
1098
1099
1100
1101
1102
1103
1104
1105
1106
1107
1108
1109
1110
1111
1112
1113
1114
1115
1116
1117
1118
1119
1120
1121
1122
1123
1124
1125
1126
1127
1128
1129
1130
1131
1132
1133
1134
1135
1136
1137
1138
1139
1140
1141
1142
1143
1144
1145
1146
1147
1148
1149
1150
1151
1152
1153
1154
1155
1156
1157
1158
1159
1160
1161
1162
1163
1164
1165
1166
1167
1168
1169
1170
1171
1172
1173
1174
1175
1176
1177
1178
1179
1180
1181
1182
1183
1184
1185
1186
1187
1188
1189
1190
1191
1192
1193
1194
1195
1196
1197
1198
1199
1200
1201
1202
1203
1204
1205
1206
1207
1208
1209
1210
1211
1212
1213
1214
1215
1216
1217
1218
1219
1220
1221
1222
1223
1224
1225
1226
1227
1228
1229
1230
1231
1232
1233
1234
1235
1236
1237
1238
1239
1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267
1268
1269
1270
1271
1272
1273
1274
1275
1276
1277
1278
1279
1280
1281
1282
1283
1284
1285
1286
1287
1288
1289
1290
1291
1292
1293
1294
1295
1296
1297
1298
1299
1300
1301
1302
1303
1304
1305
1306
1307
1308
1309
1310
1311
1312
1313
1314
1315
1316
1317
1318
1319
1320
1321
1322
1323
1324
1325
1326
1327
1328
1329
1330
1331
1332
1333
1334
1335
1336
1337
1338
1339
1340
1341
1342
1343
1344
1345
1346
1347
1348
1349
1350
1351
1352
1353
1354
1355
1356
1357
1358
1359
1360
1361
1362
1363
1364
1365
1366
1367
1368
1369
1370
1371
1372
1373
1374
1375
1376
1377
1378
1379
1380
1381
1382
1383
1384
1385
1386
1387
1388
1389
1390
1391
1392
1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508
1509
1510
1511
1512
1513
1514
1515
1516
1517
1518
1519
1520
1521
1522
1523
1524
1525
1526
1527
1528
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2

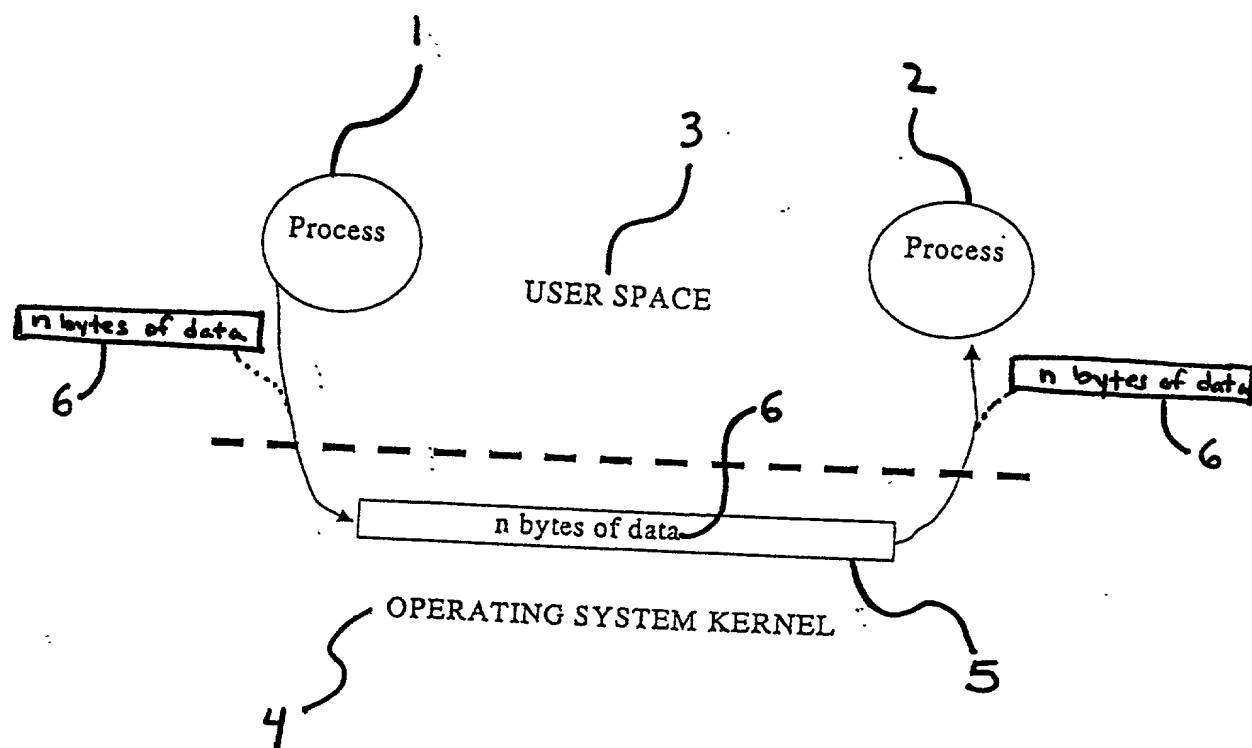


FIG. 1

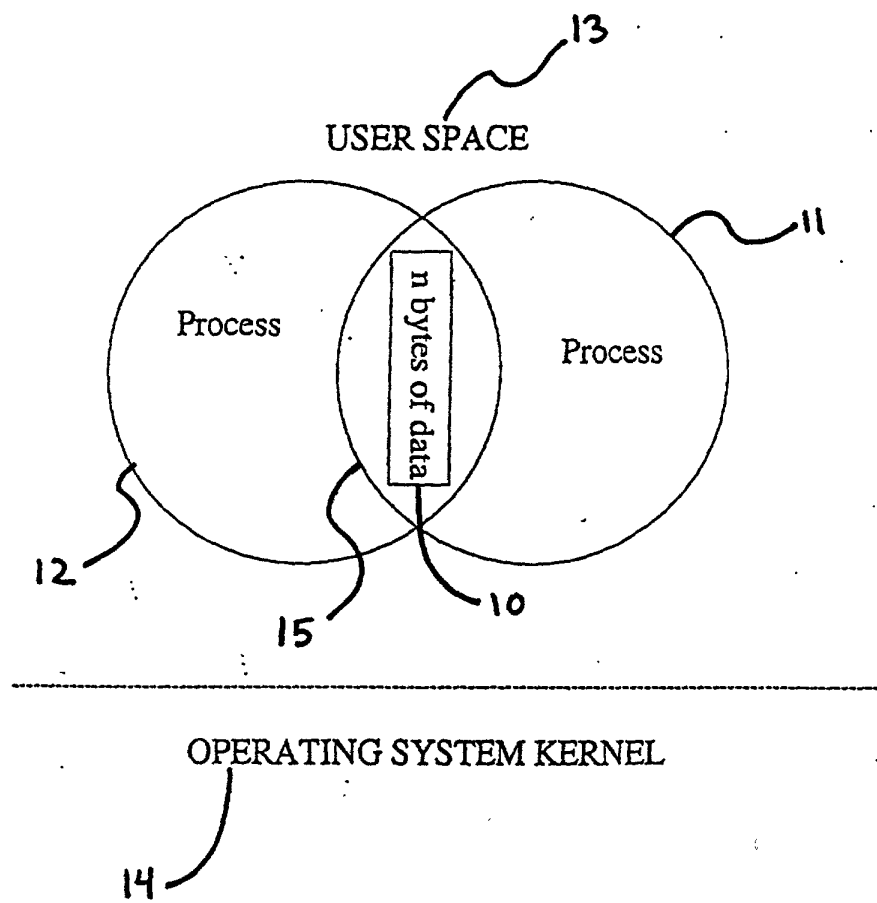


FIG. 2

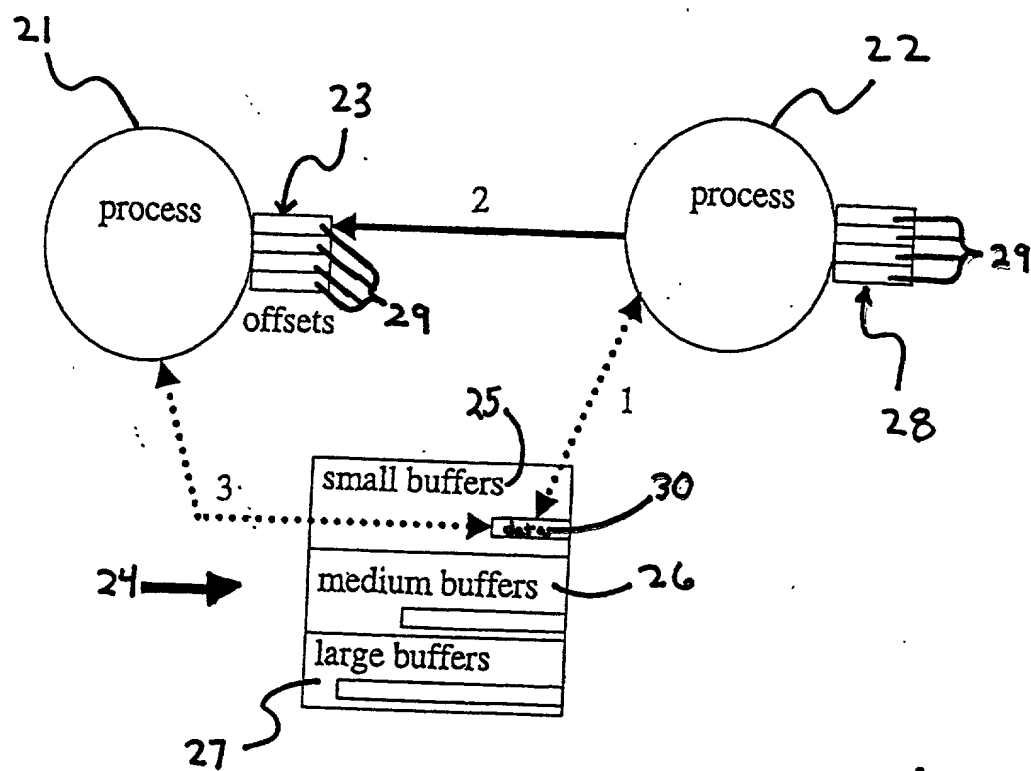
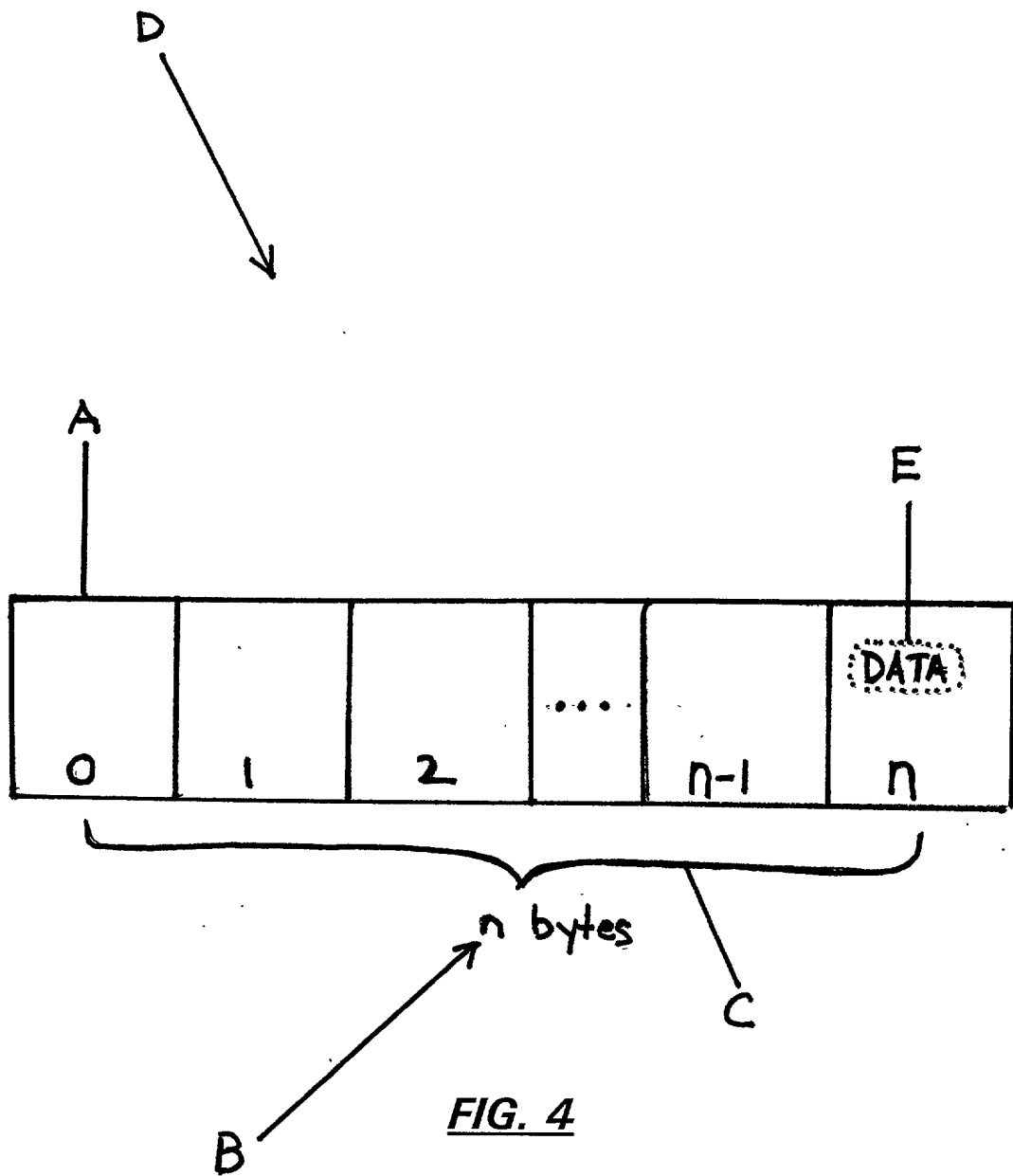


FIG. 3



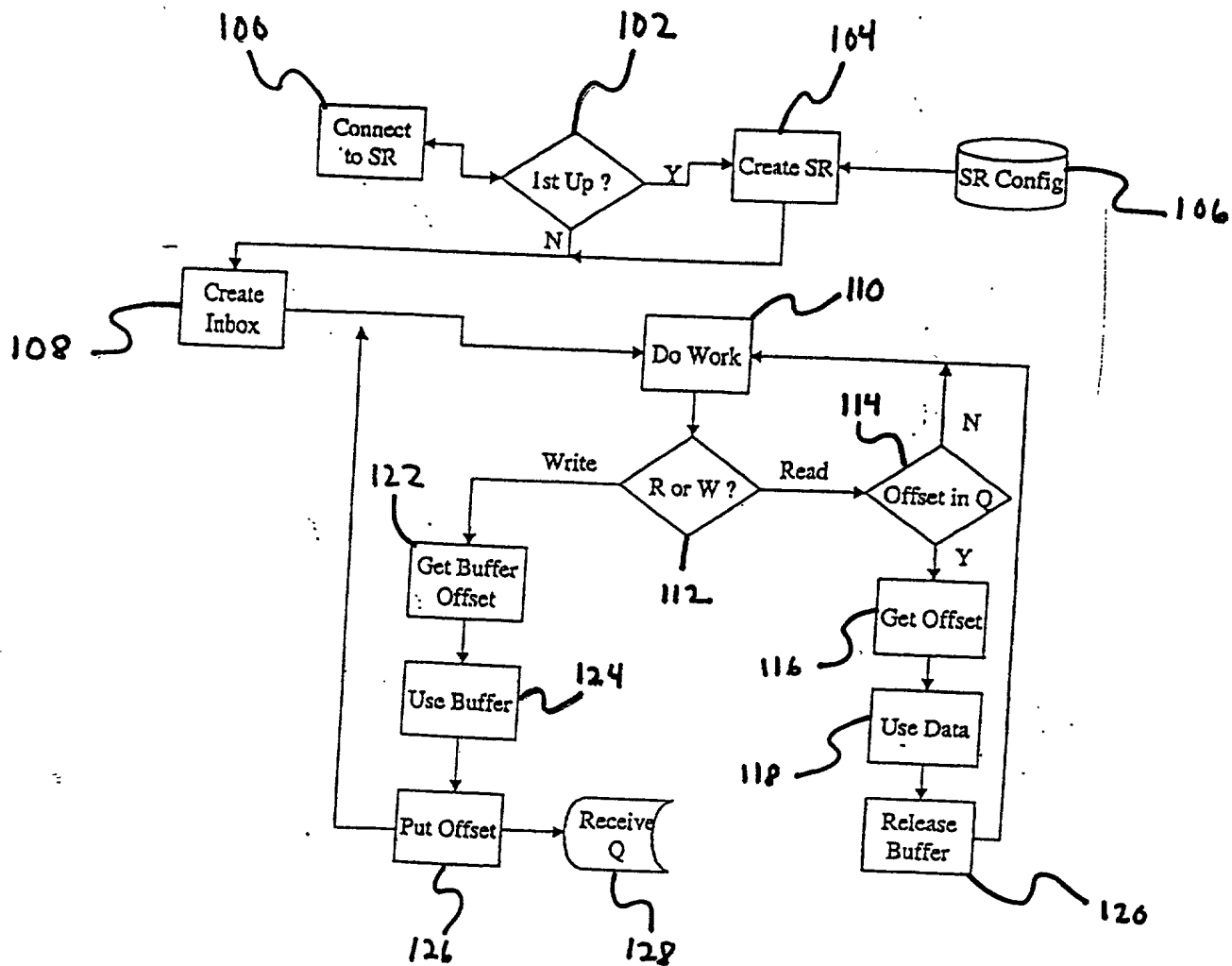


FIG. 5

DECLARATION FOR PATENT APPLICATION

Docket No: 6572-14

As a below named inventor, I hereby declare that:

My residence, post office address and citizenship are as stated below next to my name.

I believe I am the original, first and sole inventor (if only one name is listed below) or an original, first and joint inventor (if plural names are listed below) of the subject matter which is claimed and for which a patent is sought on the invention entitled HIGH SPEED INTERPROCESS COMMUNICATION

the specification of which (check one)

X is attached hereto.
_____ was filed on _____ as
Application Serial No. _____ and
was amended on _____ (if applicable).

I hereby state that I have reviewed and understand the contents of the above identified specification, including the claims, as amended by any amendment referred to above.

I acknowledge the duty to disclose information which is material to the examination of this application in accordance with Title 37, Code of Federal Regulations Section 1.56(a).

I hereby claim foreign priority benefits under Title 35, United States Code 119 of any foreign application(s) for patent or inventor's certificate listed below and have also identified below any foreign application for patent or inventor's certificate having a filing date before that of the application on which priority is claimed:

Prior Foreign Application(s)			Priority Claimed	
(Number)	(Country)	(Day/Month/Year Filed)	Yes	No

I hereby claim the benefit under Title 35, United States Code, Section 120 of any United States application(s) listed below and, insofar as the subject matter of each of the claims of this application is not disclosed in the prior United States application in the manner provided by the first paragraph of Title 35, United States Code, Section 112, I acknowledge the duty to disclose material information as defined in Title 37, Code of Federal Regulations, Section 1.56(a) which occurred between the filing date of the prior application and the national or PCT international filing date of this application:

(Appln. Serial No.)	(Filing Date)	(Status-patent, pending, abandoned)
---------------------	---------------	-------------------------------------

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

QBWPB\149625.1

EXP.MAIL:EE444322314US

I hereby appoint J. Rodman Steele, Jr., Registration No. 25,931; Gregory A. Nelson, Registration No. 30,577; Joseph W. Bain, Registration No. 34,290; Robert J. Sacco, Registration No. 35,667; Mark D. Passler, Registration No. 40,764; Stanley A. Kim, Registration No. 42,730; Steven M. Greenberg, Registration No. 44,725; and Scott D. Paul, Registration No. 42,984 as my attorneys with full power of substitution and revocation to prosecute this application and to transact all business in the Patent and Trademark Office connected therewith, to amend the specification, to appeal in case of rejection, as they may deem advisable, to receive the patent when granted and generally to do all matters and things needful in the premises as fully and to all intents and purposes as I could do.

Please direct all correspondence to: ROBERT J. SACCO

QUARLES & BRADY LLP
222 Lakeview Avenue - Suite 400
P. O. Box 3188
West Palm Beach, Florida 33402-3188
Telephone: (561) 653-5000

Full name of sole or first inventor Anthony P. Galluscio

Inventor's signature _____ Date _____

Residence Indialantic, Florida

Citizenship USA

Post Office Address 549 Humming Bird Drive, Indialantic, FL 32903

Full name of second joint inventor, if any William L. Holt

Second Inventor's signature _____ Date _____

Residence Coco Beach, Florida

Citizenship USA

Post Office Address 57 S. Atlantic Avenue, Coco Beach, Florida 32931

Full name of third joint inventor, if any Douglas M. DYER

Second Inventor's signature _____ Date _____

Residence Indialantic, Florida

Citizenship USA

Post Office Address 135 Eighth Avenue, Indialantic, Florida 32903

Post Office Address 4260 Windover Way, Melbourne, Florida 32934